USING GENETIC ALGORITHMS AND TABU SEARCH PARALLEL MODELS TO SOLVE THE SCHEDULING PROBLEM

Pedro Pinacho*
ppinacho@diinf.usach.cl

Mauricio Solar*
msolar@usach.cl

Mario Inostroza*
minostro@diinf.usach.cl

Rosa Muñoz*
rmunoz@diinf.usach.cl
(*) Departamento de Ingeniería Informática,
Universidad de Santiago de Chile,
Av. Ecuador 3659 Santiago, Chile.

Abstract

This work presents a comparison between a Parallel Genetic Algorithm (PGA) and a Parallel Tabu Search (PTS) algorithm. Both are used for solving a scheduling problem on 45 tests based on the same parallel model, (Synchronous Network Concurrency Model) and testing sequential algorithms with 2, 4 and 8 processors. The results show that the PTS algorithm obtains better results, and covers a smaller portion of the solution space.

Keywords: Genetic Algorithms, Tabu Search, Parallelism, Scheduling.

Introduction

One of the problems evidenced in an application that requires the computational power of a parallel machine is the assignment of tasks

to the processors available with the purpose of reducing the total computational time utilized. This problem, referred to an instance of the *scheduling* problem, is represented by weighted Directed Acyclic Graphs (DAGs) also known as task graphs which configure a set of tasks in homogenous processors in order to minimize total required time. In its general form this is a NP-complete problem although there are polynomial solutions known for a few restricted cases [3], [7].

The representation of the tasks and assignment in a *DAG* [8], in these graphs is defined by the tuple $G = (V,E,C,T)$, where $V = \{n_j, j = 1 : v\}$ is the set of task nodes and $v = |V|$ is the number of nodes, E is the set of communication arcs, $e = |E|$ is the number of arcs, C is the cost of the communication arcs and T is the set of computation time of the nodes. The value $c_{i,j} \in C$ is the time of communication incurred throughout the arcs $e_{ij} = (n_i, n_j) \in E$, which is zero if both arcs are assigned to the same processor. The value $t_i \in T$ is the computation time of the node $n_i \in V$.

A task is an indivisible computation unit which may be an instruction, routine or an entire program. Additionally a task is considered non-preemtive due to the fact that once it is begun it must be completed without interruptions.

This article presents the context of the general scheduling problem in Section 2, defining the type of problem encountered, later in Section 3 the metaheuristics used are described. Section 4 shows how GA and TS have been parallelized, while Section 5 presents the tests carried out and the analysis of these. Finally, Section 6 presents the conclusions reached in this work.

1. The Problem

The scheduling problem is defined as follows: given a number v of tasks (where t_i corresponds to the computation time of the i-th task) which must be carried out by P processors (where p_j corresponds to the j-th processor), it is required to know which tasks must be carried out on which processor, such that the total time to completion be as small as possible.

A few instances of the problem are described below:

Dependence among tasks: In order that task i begin its execution, tasks $t_1, t_2, ..., t_k$, must have concluded, that is, a certain established order must be respected (where k is the quantity of tasks that precede task i). In this respect, there are two possibilities:

- There is dependency between tasks.

- No task depends on the execution of another.

Duration of the task: This refers to the duration of the tasks. Two cases exist:

- All tasks have the same duration*(unitary)*.
- The tasks have different and arbitrary duration.

Communication Cost: Corresponds to the time required to communicate one task t_k with its predecessors whose execution has finished. With respect to this, there are two different cases:

- Task t_i must communicate with task t_k, to be carried out: if both tasks are executed on the same processor, the cost is considered to be zero or insignificant; if they are carried out on different processors there is a known associated cost (c_{ij}).
- Communication costs are not considered.

Number of Processors: Corresponds to the quantity of available resources to resolve a given problem. There are two alternatives:

- An unbounded number of processors is considered.
- The number of processors (P) available for the problem is specified.

Processor Homogeneity: This has to do with the specific characteristics of the processors. There are two cases:

- The processors are identical.
- The processors do not have the same characteristics.

In the present work, the scheduling problem considered has the following characteristics (instance):

- There is dependency among tasks.
- Tasks differ in duration (arbitrary).
- Communication cost between tasks is taken into consideration.
- Processors are homogenous and limited.

Researchers in this area of study have devoted approximately forty years to develop various algorithms for this problem. A real alternative are Metaheuristic Algorithms. These are generally defined, and thus it is

necessary to define their parameters in order to model the problem to be solved according to its nature. Their principal characteristic is that they do not ensure the finding of the optimal solution, but rather solutions that are close to the optimum. Some examples of these are Simulated Annealing [11], Genetic Algorithms [10], and Tabu Search [9].

2. Tabu Search and Genetic Algorithms

This work centers on the study of the effectiveness of the techniques of Genetic Algorithms and Tabu Search for the solution of a specific instance of the scheduling problem which are presented below.

2.1 Tabu Search (TS)

Tabu Search is a metaheuristic method introduced and developed in its present form by [9]. In General TS consists of generating, on the basis of a random and feasible initial solution, neighboring solutions from which the optimal solution is selected (or not) when compared to the initial solution. A Tabu List (L) is generated of the movements that are not allowed in the present iteration, (in this way movements that could result in the selection of a local optimum are excluded). There are different criteria for determining when it is possible to remove a movement from L. Intensification and diversification strategies define how close the generated solutions are to the initial solution.

The modeling used for this problem coincides with genetic algorithms in representation of individuals and in the evaluation function. The generation of a neighboring solution consists of changes in the positions of the elements of the initial solution. The criteria for functioning for a neighborhood were aspiration by default [9]. The search strategy is defined by the following parameters presented in [4]:

1 The size of L is 100.

2 The number of variations permitted is the average of the DAG input tasks, and the number of processors.

3 The size of the neighborhood is the sum of the previous.

When the number of iterations allowed in each cycle is completed, the search strategy is updated, and the number of differences between solutions is analyzed (Hamming distance). If it is larger than 50% of the size of a solution, the size of L is reduced as are the number of variations, and the size of the neighborhood is increased; if it is smaller, the size of L is increased as is the number of variations, and the size of the neighborhood is decreased.

2.2 Genetic Algorithms (GA)

Genetic Algorithms are metaheuristic methods [10], (of an evolutionary nature) that belong to the area of Artificial Intelligence, and make it possible to find solutions that are close to optimum for difficult optimization problems. GAs base their search on the mechanics of natural selection and genetics, where evolution operates directly on the chromosomes of living organisms by means of selection processes, crosses between individuals (thus generating offspring that are different to the parents), and mutations which allows adaptation to changes in the environment. According to this idea, GAs use an initial population of individuals (possible solutions for the modeled problem) that is random, finite and same size for each generation. The individuals are represented by binary strings in which a 1 represents a characteristic present in the chromosome of the individual (pure GAs). The initial population evolves from one generation to the next a fixed number of times by means of genetic operators. These select the best solutions according to a defined evaluation function, reproduce the individuals randomly changing the genetic information in two chromosomes, thus generating new individualism, and mutating some characteristic of the chromosome.

The model used in this work [12], corresponds to a hybrid representation of the chromosomes where the length of an individual is equal to the number of tasks plus the number of available processors. Each characteristic has a positive number between 1 and the number of tasks, (which represent the scheduled task in the DAG) or a negative number between 1 and the number of processors, (which represents the number of the processor where the assignments are being carried out). Reading the chromosome from left to right, the positive numbers next to negative numbers indicate what tasks are to be carried out, and in what order (Figure 1).

The evaluation function associated to this model determines the time that each individual requires per task, taking into consideration the associated computation and communication costs. The operators used in this modeling exercise as justified in [12], and whose parameters where tuned after a process of adjustment, are the following [6]:

Mutation: Two random positions for the individual are chosen and exchanged. Percent mutation: 1%.

Crossover: Elements from one of the parents are randomly selected q and are stored in a row. Offsprings are created by copying each element from the other parent from left to right, if the element

is in the row, the following element is copied. Percent crossover: 30%.

Selection: Individuals are selected randomly from a population, those that achieve the highest score by the evaluation function, have a higher probability of being selected. Selection percentage: 50%.

The size of the population in each generation is of 400 individuals.

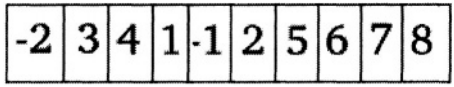


Figure 1. Chromosome representation of the GA

3. Parallel Tabu Search and Parallel Genetic Algorithms

Just as it is interesting to study the scheduling problem due to its application in the optimization of a parallel machine's resources, it is also interesting to apply parallel models to Meta-Heuristic techniques to extrapolate the improvements in process time allowed by parallelism. There are a variety of applicable parallel models [5], and different studies on the parallelization of specific methods [1]. Consequently, there is justification for the comparison of the parallel behavior between two known methods such as Genetic Algorithms [6], and Tabu Search [4].

3.1 Synchronous Network Concurrency Model

The parallelization of the metaheuristics used in this work is achieved through the *Synchronous Network Concurrency Model.* This parallel model uses K processes executed independently, and with independent information. The best result is communicated to a master process which determines the best solution and communicates this to the concurrent processes [13].

The model uses coarse grain parallelization [1], in which the population is divided into sub-populations which are kept relatively isolated from each other. This model introduces the migration operator, which is used to send individuals of a sub-population to the master.

The most frequently used population models in the implementation of coarse grained genetic algorithms are the following: The *island* model, and the *stepping stone* model. In the *island* model the population is

sub-partitioned into geographically isolated sub-populations and the individuals can migrate to any other sub-population. In the *stepping stone* model the population is partitioned in the same way, however migration is restricted to neighboring populations.

3.2 Parallel Genetic Algorithm *(PGA)*

The model used consists of a master GA and *K* independent GAs (see Figure 2). The population is divided into equal parts for each GA. When one generation is finished, the GAs send their best individual to the master GA which determines the best of these sending it on to all the GAs according to the migration policy *best individual over random individual* (The best individual is copied over any solution for every GA [13]).

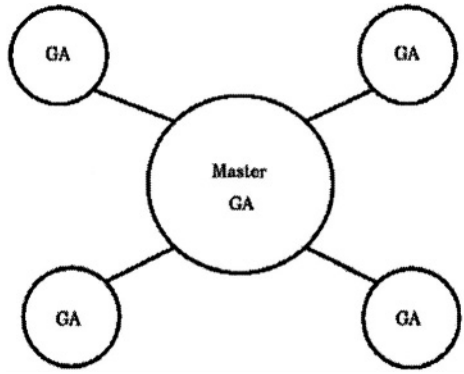


Figure 2. Synchronous Network Concurrency Model applied to Parallel Genetic Algorithms

3.3 Parallel Tabu Search *(PTS)*

As with the parallel GA described above, Tabu Search has *K* TSs with independent neighborhoods and tabu lists. Each sends its best solution to the master TS which copies the best of these to the rest of the TSs. The master process also takes care of initializing the parameters for the search strategy and the initial solutions for each TS.

4. Tests Carried Out and Analysis of Results

The DAG test set presented is made up of 15 representative DAGs extracted from the study [14], and whose principal characteristics are detailed in Table 1. The scheduling problem used considers the assign-

Table 1. Test DAG characteristics

DAG	v	e	σt_i	$\sigma c_{i,j}$	Type of DAG
dag0000	8	8	8	8	Unitary/Unitary
dag0001	9	9	90	23	Arbitrary/Arbitrary
dag0002	9	9	9	23	Unitary/Arbitrary
dag0003	10	9	43	22	Arbitrary/Arbitrary
dag0004	8	9	8	9	Unitary/Unitary
dag0007	9	12	9	12	Unitary/Unitary
dag0008	10	9	10	22	Unitary/Arbitrary
dag0009	20	19	85	52	Arbitrary/Arbitrary
dag0011	20	19	20	19	Unitary/Unitary
dag0012	21	20	21	20	Unitary/Unitary
dag0014	21	20	48	20	Arbitrary/Unitary
dag0026	14	13	14	13	Unitary/Unitary
dag0027	13	17	13	17	Unitary/Unitary
dag0029	13	18	25	64	Arbitrary/Arbitrary
dag0033	6	6	6	6	Unitary/Unitary

ment of 15 DAGs for $P = 2, 4$ and 8 (45 tests). For each test, each algorithm is executed ten times: GA and TS with $K = 1, 2, 4$ and 8.

The tests were carried out on a machine with shared memory *Silicon Graphics,* with an Irix operating system, version 4.3 and C language.

The information evaluated for each test is the following: Mean Parallel Time *(PTmean),* the size of the Mean Space Covered *(SCmean)* and the Minimum Parallel Time obtained during the 10 executions*(PTmin).* Figures 3 and 4, show the graphs of the results obtained from *PTmean* and *SCmean* for the mentioned graphs itinerated over 2 processors ($P = 2$), with 2 parallel algorithms for GA and TS ($K = 2$).

The need for metaheuristics to approach the scheduling problem is justified upon inspection of the size of the search space [4] by means of the equation 1a [12], which establishes the number of possible methods for resolving a number of tasks v over a number of processors P, while considering the dependency among tasks.

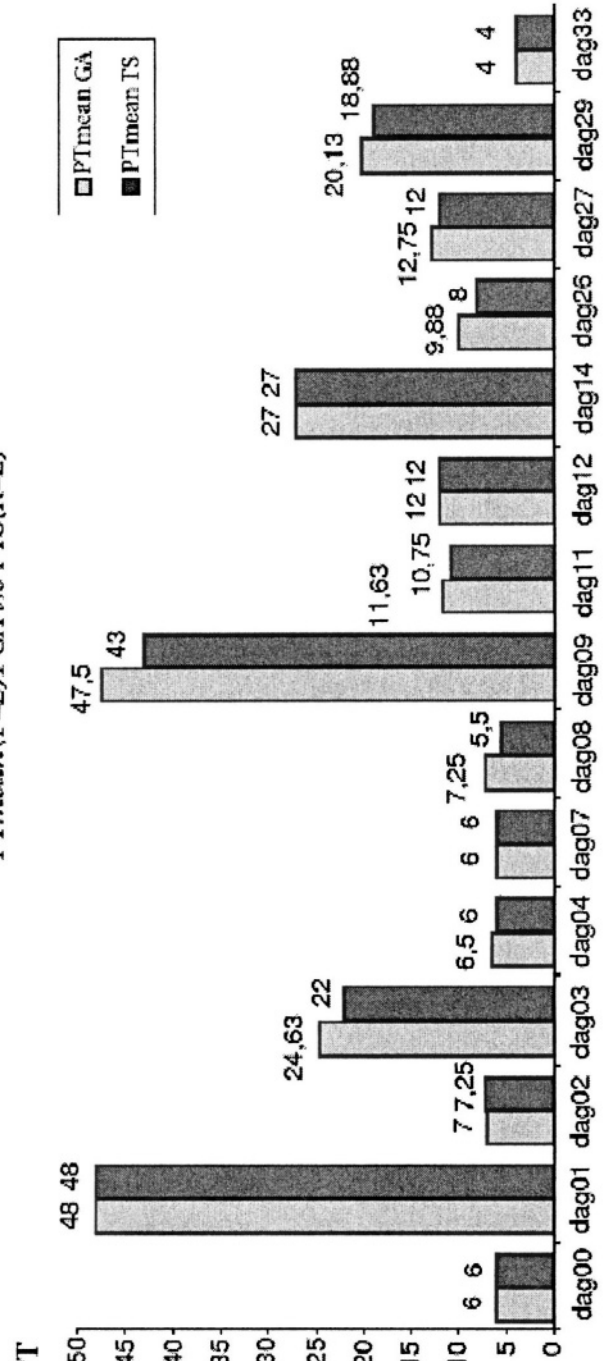
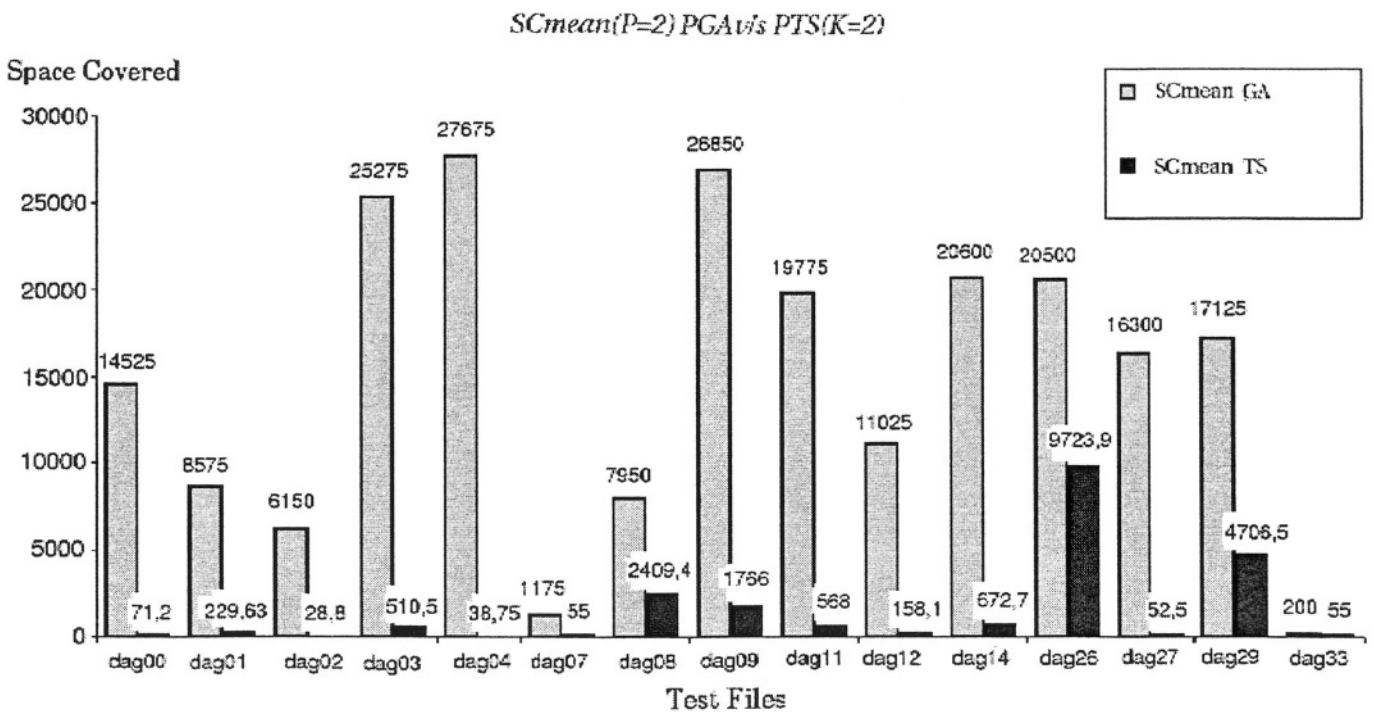


Figure 3. **Mean Parallel Time between GA and TS with $K = 2, P = 2$.**

Figure 4. Mean Space Covered between GA and TS with $K = 2, P = 2$.



$$f(v,P) = \begin{cases} 1, & \text{if } v = 0 \wedge P = 0 \\ 1, & \text{if } v = 1 \wedge P = 0 \\ P, & \text{if } v = 1 \\ v!, & \text{if } P = 1 \\ \sum_{i=0}^{v} (v-i)! f(i,(P-1)) \left| \begin{matrix} v \\ i \end{matrix} \right|, & \text{O.C.} \end{cases} \quad (1a)$$

With equation 1a table A.1 is deduced, with the sizes of the search spaces of the reviewed *DAGs*, thus highlighting the technical impossibility of obtaining solutions by exhaustive search in order to find an optimal solution because of the combinatorial explosion.

4.1 Analysis of Results

Figure 3 shows that the *Mean Parallel Time* of the TS based solutions, for scheduling with two processors ($P = 2$), are better than those achieved with the GAs. Additionally, upon reviewing the Figure 4, it can be seen that the *Mean Space Covered* for the GAs with two processors, is greater than TS.

Of the 45 tests carried out, the most representative of the behavior of the algorithms, are those carried out on the graphs assigned for $P = 8$. From the 15 tests selected, in 7 cases GA and TS obtained equal results for parallel time. Additionally, there are 4 cases in which TS obtained equal results independently of the degree of parallelism used, while the *Mean Parallel Time* increased.

As can be seen in Figure 5, *PTS* showed a better performance than *PGA* for all the configurations tested ($K = 1,2,4,8$).

In terms of the *Mean Space Covered* for the mentioned DAGs with 8 processors ($P = 8$) (Figure 6), the PGAs decrease their exploration in the measure that the degree of parallelism increases confirming the results shown in [13]. For their part the PTS show a similar performance, although for *Mean Space Covered* always less, and a slower decrease in the measure that the degree of parallelism increases.

5. Conclusions

According to the results obtained, it can be seen that the *PTS* shows better results than the *PGA*. This can be deduced from the observation that for similar quality solutions *(Mean Parallel Time),* the *Mean Space Covered* for PTS is significantly less and is in fact a fraction of that used by PGAs. In practice, there is an obvious decrease in required processor time in order to find a good solution to the problem of scheduling multiprocessor machines.

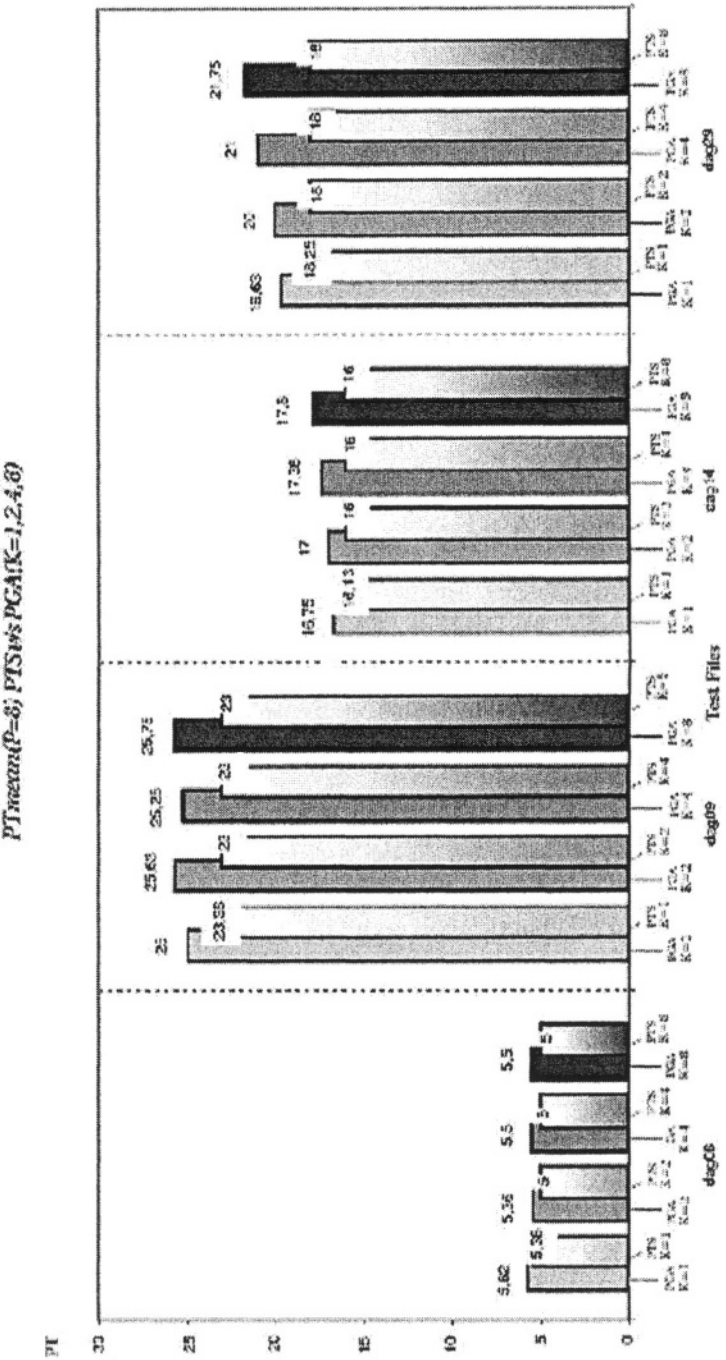


Figure 5. Mean Parallel Time between GA and TS with $K = 1, 2, 4, 8$ where $P = 8$.

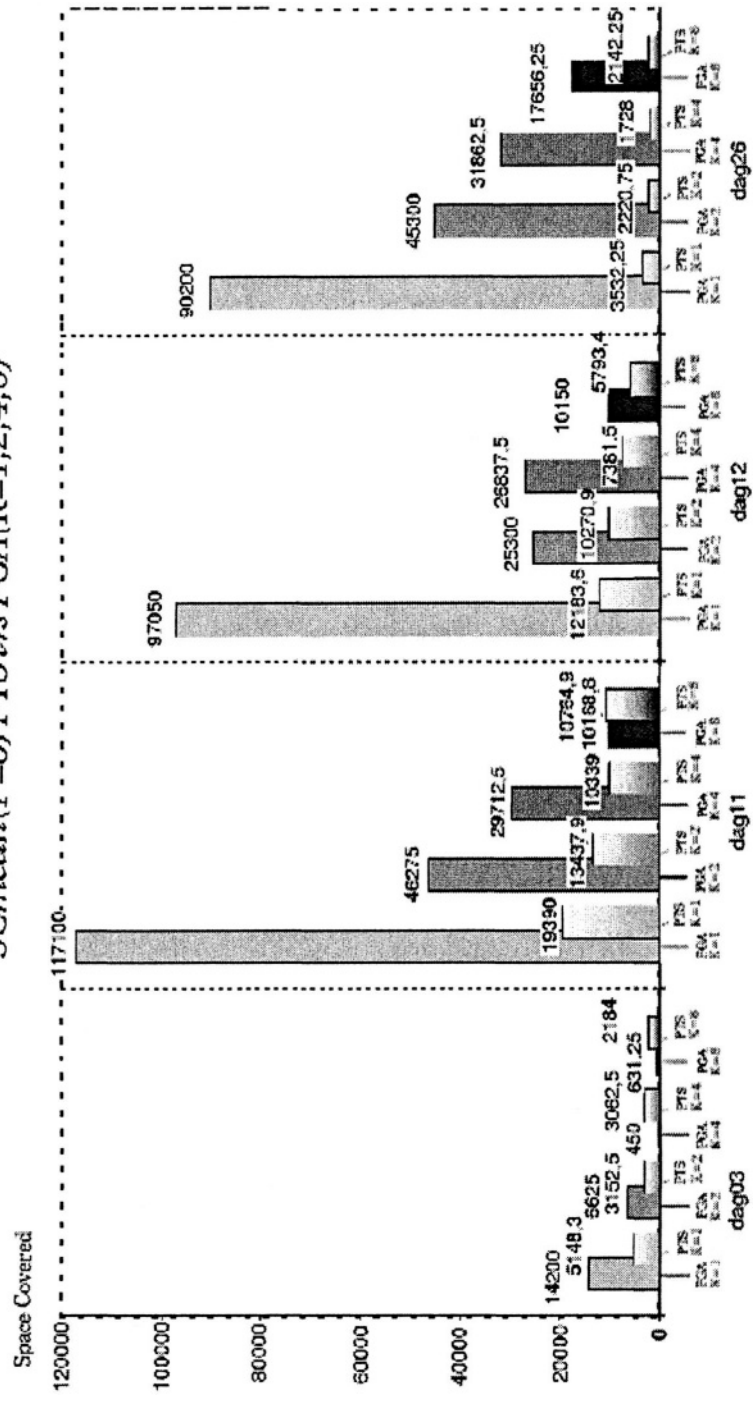


Figure 6. **Mean Space Covered between GA and TS with $K = 1, 2, 4, 8$ where $P = 8$.**

The PTS and PGA can be used to solve other instances of the scheduling problem, perhaps with similar results. As future works we are studying the application of these parallel models to solve other kinds of combinatorial problems.

Acknowledgments

This work was partially supported by FONDECYT 1030775.

Appendix

See Table A.1 on next page.

References

[1] Cantu Paz E. Summary of Research on Parallel Genetic Algorithms. Illinois Genetic Algorithms Laboratory, USA, 1995.

[2] Cantu Paz E. Migration Policies and Takeover Times in Parallel Genetic Algorithms, Illinois Genetic Algorithms Laboratory, USA, 1999.

[3] Coffman E. and Graham R. Optimal Scheduling for Two-Processor Systems. *Acta Informática,* vol.1, pp 200-213, 1972.

[4] Contreras P. Parallel Tabu Search for solving the Scheduling Problem. Final Project in Computer Engineering [in Spanish]. Universidad de Santiago de Chile, 2002.

[5] Crainic T. and Toulouse M. Parallel Metaheuristics. Centre Recherche Sur les Transport, University of Montreal, QC, Canada, 1997.

[6] Díaz S. Parallel Genetic Algorithm for solving the Scheduling Problem. Final Project in Computer Engineering [in Spanish]. Universidad de Santiago de Chile, 2002.

[7] Fernandez E. and Bussell B. Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules. *IEEE Trans. Computers,* vol C-22, no. 8, pp. 745-751, Aug. 1973.

[8] Gerasoulis A. and Tang T. A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors. *Journal on Parallel and Distributed Computing,* Vol. 16, N 4., 1992.

[9] Glover F. and Laguna M. *Tabu Search,* Kluwer Academic Publisher, 1997.

[10] Golberg D. *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, 1989.

[11] Kirkpatrick S. and Gelatt C. and Vecchi M. Optimization by Simulated Annealing, *Science,* N. 220, pp. 671-680, 1983.

[12] Kri F. Parallel Models for Genetic Algorithms with Distributed Memory. Master Thesis [in Spanish], Universidad de Santiago de Chile, 1996.

[13] Solar M. and Parada V. A parallel genetic algorithm to solve the set-covering problem, *Computer & Operation Research,* 29(1), pp. 1221-1235, November 2002.

[14] Waseda. Kasahara Laboratory, Department of Electrical, Electronics and Computer Engineering, Waseda university, http://www.kasahara.elec.waseda.ac.jp/schedule/, 2002.

Table A.1. Test DAG characteristics

				SCmean							
				GA				*TS*			
DAG	v	P	*Solution*	K=1	K=2	K=4	K=8	K=1	K=2	K=4	K=8
0	8	2	362880.0	7.2	4.0	3.4	2.5	$1.1*10^{-2}$	$2.0*10^{-2}$	$1.7*10^{-2}$	$1.4*10^{-3}$
1	9	2	$3.6*10^{6}$	0.6	$2.4*10^{-1}$	$1.3*10^{-1}$	5.010^{-2}	$1.5*10^{-2}$	$6.3*10^{-3}$	$1.1*10^{-2}$	$7.4*10^{-3}$
2	9	2	$3.6*10^{6}$	0.2	0.2	0.2	0.1	$2.0*10^{-3}$	$7.9*10^{-4}$	$2.0*10^{-3}$	$2.2*10^{-3}$
3	10	2	$4.0*10^{7}$	0.1	0.1	$4.0*10^{-2}$	$2.0*10^{-3}$	$2.4*10^{-3}$	$1.0*10^{-3}$	$8.3*10^{-4}$	$4.4*10^{-4}$
3	10	8	$7.*10^{10}$	$0.2*10^{-4}$	$9.4*10^{-6}$	$6.4*10^{-7}$	$8.9*10^{-7}$	$7.2*10^{-6}$	$4.5*10^{-6}$	$4.3*10^{-6}$	$3.1*10^{-6}$
4	8	2	$3.6*10^{5}$	10.9	7.6	2.4	2.5	0.1	$0.1*10^{-1}$	$9.3*10^{-3}$	$0.2*10^{-1}$
7	9	2	$3.6*10^{6}$	0.1	$0.3*10^{-1}$	$0.2*10^{-1}$	$0.1*10^{-1}$	$1.6*10^{-3}$	$1.5*10^{-3}$	$3.2*10^{-3}$	$3.4*10^{-3}$
8	10	2	$4.0*10^{7}$	$4.1*10^{-2}$	$2.0*10^{-2}$	$1.6*10^{-2}$	$9.7*10^{-3}$	$3.3*10^{-3}$	$26.1*10^{-3}$	$6.8*10^{-3}$	$4.0*10^{-3}$
8	10	8	$7.1*10^{10}$	$1.4*10^{-4}$	$7.9*10^{-5}$	$1.1*10^{-5}$	$8.5*10^{-6}$	$9.1*10^{-6}$	$9.2*10^{-6}$	$9.4*10^{-6}$	$5.5*10^{-6}$
9	20	2	$5.1*10^{19}$	$1.1*10^{-13}$	$5.3*10^{-14}$	$2.7*10^{-14}$	$1.6*10^{-14}$	$5.9*10^{-15}$	$3.4*10^{-15}$	$2.4*10^{-15}$	$2.4*10^{-15}$
9	20	8	$2.2*10^{24}$	$8.1*10^{-18}$	$4.4*10^{-18}$	$1.7*10^{-18}$	$1.1*10^{-18}$	$8.6*10^{-19}$	$8.7*10^{-19}$	$6.8*10^{-19}$	$5.3*10^{-19}$
11	20	2	$5.1*10^{19}$	$1.0*10^{-13}$	$3.9*10^{-14}$	$1.7*10^{-14}$	$8.9*10^{-15}$	$3.3*10^{-15}$	$1.1*10^{-15}$	$2.3*10^{-15}$	$3.2*10^{-15}$
11	20	8	$2.2*10^{24}$	$5.4*10^{-18}$	$2.1*10^{-18}$	$1.4*10^{-18}$	$4.7*10^{-19}$	$9.0*10^{-19}$	$6.2*10^{-19}$	$4.8*10^{-19}$	$5.0*10^{-19}$
12	21	2	$1.1*10^{21}$	$2.2*10^{-15}$	$9.8*10^{-16}$	$7.0*10^{-16}$	$3.9*10^{-16}$	$2.7*10^{-17}$	$1.4*10^{-17}$	$1.0*10^{-17}$	$1.6*10^{-17}$
12	21	8	$6.0*10^{25}$	$1.6*10^{-19}$	$4.1*10^{-20}$	$4.4*10^{-20}$	$1.7*10^{-20}$	$2.0*10^{-20}$	$1.7*10^{-20}$	$1.2*10^{-20}$	$9.6*10^{-21}$
14	21	2	$1.1*10^{21}$	$2.0*10^{-15}$	$1.8*10^{-15}$	$7.8*10^{-16}$	$5.3*10^{-16}$	$4.7*10^{-17}$	$6.0*10^{-17}$	$4.4*10^{-17}$	$6.5*10^{-17}$
14	21	8	$6.0*10^{25}$	$1.9*10^{-19}$	$1.1*10^{-19}$	$4.2*10^{-20}$	$3.3*10^{-20}$	$2.8*10^{-20}$	$2.1*10^{-20}$	$1.7*10^{-20}$	$1.1*10^{-20}$
26	14	2	$1.3*10^{12}$	$2.4*10^{-6}$	$1.6*10^{-6}$	$7.5*10^{-7}$	$4.0*10^{-7}$	$3.4*10^{-7}$	$7.4*10^{-7}$	$4.0*10^{-7}$	$2.0*10^{-7}$
26	14	8	$1.0*10^{16}$	$8.9*10^{-10}$	$1.2*10^{-10}$	$3.1*10^{-10}$	$1.7*10^{-10}$	$3.5*10^{-11}$	$2.2*10^{-11}$	$1.7*10^{-11}$	$2.1*10^{-11}$
27	13	2	$8.7*10^{10}$	$2.8*10^{-5}$	$1.9*10^{-5}$	$9.3*10^{-6}$	$5.9*10^{-6}$	$6.7*10^{-8}$	$6.0*10^{-8}$	$6,5*10^{-8}$	$1.2*10^{-7}$
27	13	8	$4.8*10^{14}$	$1.3*10^{-8}$	$6.4*10^{-9}$	$3.8*10^{-9}$	$2.8*10^{-9}$	$5.7*10^{-11}$	$3.3*10^{-11}$	$3.4*10^{-11}$	$7.2*10^{-11}$
29	13	2	$8.7*10^{10}$	$2.6*10^{-5}$	$2.0*10^{-5}$	$1.3*10^{-5}$	$5*10^{-6}$	$1.4*10^{-6}$	$5.4*10^{-6}$	$1.3*10^{-6}$	$1.9*10^{-6}$
29	13	8	$4.8*10^{14}$	$3.6*10^{-8}$	$1.4*10^{-8}$	$1.2*10^{-8}$	$3.0*10^{-9}$	$1.1*10^{-9}$	$1.1*10^{-9}$	$10.0*10^{-10}$	$7.4*10^{-10}$
33	6	2	$5.0*10^{3}$	7.9	4.0	2.0	1.0	1.1	1.1	1.9	2.0